Tight conditional lower bounds for counting perfect matchings on graphs of bounded treewidth, cliquewidth, and genus

Radu Curticapean* Dániel Marx†

October 14, 2015

Abstract

By now, we have a good understanding of how NP-hard problems become easier on graphs of bounded treewidth and bounded cliquewidth: for various problems, matching upper bounds and conditional lower bounds describe exactly how the running time has to depend on treewidth or cliquewidth. In particular, Fomin et al. (2009, 2010) have shown a significant difference between these two parameters: assuming the Exponential-Time Hypothesis (ETH), the optimal algorithms for problems such as MAX CUT and EDGE DOMINATING SET have running time $2^{O(k)}n^{O(1)}$ when parameterized by treewidth, but $n^{O(1)}$ when parameterized by cliquewidth.

In this paper, we show that a similar phenomenon occurs also for counting problems. Specifically, we prove that, assuming the counting version of the Strong Exponential-Time Hypothesis ($\#$ETH), the problem of counting perfect matchings

- has no $(2 - \varepsilon)^kn^{O(1)}$ time algorithm for any $\varepsilon > 0$ on graphs of treewidth $k$ (but it is known to be solvable in time $2^{k}n^{O(1)}$ if a tree decomposition of width $k$ is given), and
- has no $O(n^{1-\varepsilon}k)$ time algorithm for any $\varepsilon > 0$ on graphs of cliquewidth $k$ (but it can be solved in time $O(n^{k+1})$ if a $k$-expression is given).

A celebrated result of Fisher, Kasteleyn, and Temperley from the 1960s shows that counting perfect matchings in planar graphs is polynomial-time solvable. This was later extended by Gallucio and Loebl (1999), Tesler (2000) and Regge and Zechina (2000) who gave $4^{k} \cdot n^{O(1)}$ time algorithms for graphs of genus $k$. We show that the dependence on the genus $k$ has to be exponential: assuming $\#$ETH, the counting version of ETH, there is no $2^{O(k)} \cdot n^{O(1)}$ time algorithm for the problem on graphs of genus $k$.

1 Introduction

Many NP-hard optimization problems are solvable in polynomial time when restricted to graphs of bounded treewidth. This fact is exploited in a wide range of contexts, perhaps most notably, in the design of parameterized algorithms and approximation schemes. There has been a significant amount of research on developing and improving algorithms for bounded-treewidth graphs, as well as on trying to understand the limitations of treewidth-based algorithms. Courcelle’s Theorem [25, 26] is a very general result showing that if a problem can be formulated in a logical language called MSO$_2$, then it can be solved in linear-time on graphs of bounded treewidth; however, problem-specific techniques usually give more efficient algorithms that have better dependence on treewidth.

Thanks to a series of recent results, by now we have a fairly good understanding on how the running time has to depend on the treewidth of the input graph. Concerning upper bounds, the development of new algorithmic techniques, such as fast subset convolution [6, 88], Cut & Count [36, 76], and rank-based dynamic programming [8, 34, 53], improved the running times for various problems, sometimes in unexpected ways. On the other hand, matching conditional lower bounds were obtained for many of these problems [9, 12, 35, 36, 71, 72]. For example, fast subset convolution can be used to solve DOMINATING SET in time $3^{k}n^{O(1)}$ if a tree decomposition of width $k$ is given [88], but there is no such algorithm with running time $(3 - \varepsilon)^{k}n^{O(1)}$ for any $\varepsilon > 0$, assuming the Strong Exponential-Time Hypothesis (SETH) [71]. This hypothesis, introduced by Impagliazzo, Paturi, and Zane [61, 62], can be informally stated as the assertion that $n$-variable $m$-clause CNF-SAT cannot be solved in time $(2 - \varepsilon)^{m}n^{O(1)}$ for any $\varepsilon > 0$. While the validity of this hypothesis is not accepted as widely in the community as some other complexity conjectures, SETH seems to be a very fruitful working assumption that explains why certain well-known algorithms are best possible and cannot be improved further [1–4, 10, 11, 15, 33, 71, 75, 90].

---

*Simons Institute for the Theory of Computing, Berkeley, and Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary, curticapean@cs.uni-sb.de
†Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), dmarx@cs.bme.hu. Research supported by ERC Grant PARAMTIGHT (No. 280152) and OTKA grant NK105645.
Cliquewidth. The notion of cliquewidth was introduced by Courcelle and Olariu [29] and can be seen as a generalization of treewidth that keeps some of the favorable algorithmic properties of bounded-treewidth graphs. The main motivation for this width measure lies in the observation that highly homogeneous structures of large treewidth, such as cliques and complete bipartite graphs, do not pose great difficulties for, e.g., Independent Set, Dominating Set, or Vertex Coloring. This homogeneity is then captured by so-called k-expressions: For a graph G, this is a construction scheme that successively builds G from graphs whose vertices are labeled by 1, ..., k such that vertices of the same label cannot be distinguished in later steps. A graph has cliquewidth at most k if it has a k-expression.

Courcelle, Makowsky, and Rotics [27] generalized, to some extent, Courcelle’s Theorem to bounded-cliquewidth graphs, but this generalization comes at a price: it gives linear-time algorithms only for problems defined by MSO1 formulas, which is a proper subset of MSO2 that does not allow quantification over edge sets. Many problems, such as Hamiltonian Cycle, are (provably) not definable in MSO1 and there is a large literature on designing algorithms for problems on bounded-cliquewidth graphs with problem-specific approaches [48, 56–58, 67, 68, 73, 79, 81, 89]. Unlike for problems parameterized by treewidth, most of these problems are not known to be fixed-parameter tractable parameterized by cliquewidth, that is, they can be solved in time n^{f(k)}, but not in time f(k)n^{O(1)}. Fomin et al. [50, 51] showed conditional lower bounds suggesting that this is not a shortcoming of algorithm design, but an inevitable price one has to pay for the generalization to cliquewidth.

Counting perfect matchings. Some of the algorithmic results mentioned above can be generalized to the counting versions of the problems, where the task is to count the number of solutions. For example, Courcelle’s Theorem and its variant for cliquewidth have counting analogs [28]. However, it is a well-known fact that a counting problem can be significantly harder than its decision version: finding a perfect matching is a classic polynomial-time solvable problem [47], but a seminal result of Valiant showed that counting the number of perfect matchings is #P-hard [85], even in bipartite graphs, where this problem is equivalent to evaluating the permanent of a matrix. In the present paper, we show that problems such as counting perfect matchings are also amenable to the study of quantitative lower bounds outlined in the previous paragraphs: We obtain tight upper and lower bounds on the running time needed to count perfect matchings when parameterized by treewidth, cliquewidth, or genus.

Parameterizing by treewidth. The counting version of Courcelle’s Theorem [28] immediately shows that the problem #PERFMATCH of counting perfect matchings is fixed-parameter tractable parameterized by treewidth. Furthermore, standard dynamic programming techniques directly give a 3^{n^{O(1)}} time algorithm. The base of the exponential part was improved by van Rooij et al. [88] using the technique of fast subset convolution:

\textbf{Theorem 1.1.} ([88]) The problem #PERFMatch can be solved in time \(2^k n^{O(1)}\) on an n-vertex graph G if a tree decomposition of width k is given.

Our first result gives a matching lower bound: assuming #SETH (the natural counting analog of SETH, which a priori is a weaker hypothesis than SETH), the base of the exponent cannot be improved any further.

\textbf{Theorem 1.2.} Assuming #SETH, there is no algorithm that, given an n-vertex graph G together with a tree decomposition of width k, solves #PERFMATCH in time \((2-\varepsilon)^k n^{O(1)}\) for some fixed \(\varepsilon > 0\).

Our proof in fact gives a lower bound for the problem parameterized by cutwidth, a parameter that is an upper bound on treewidth and also on the related notion of pathwidth. Therefore, our proof implies Theorem 1.2, even when treewidth is replaced by pathwidth.

Parameterizing by cliquewidth. Besides the general algorithmic result on counting problems definable in MSO1, counting problems for graphs of bounded cliquewidth were investigated mostly in the context of computing graph polynomials [57, 58, 73]. Makowsky et al. [73] showed that, given a k-expression of an n-vertex graph G, the matching polynomial can be computed in time \(O(n^{2k+1})\); in particular, this gives an \(O(n^{2k+1})\) time algorithm for #PERFMATCH. In the full version of this paper, we improve this algorithm:

\textbf{Theorem 1.3.} The problem #PERFMatch can be solved in time \(O(n^{k+1})\) on an n-vertex graph G if a k-expression of G is given in the input.

Makowsky et al. [73] asked whether #PERFMATCH (or computing the matching polynomial) is fixed-parameter tractable parameterized by cliquewidth. We show that, assuming #SETH, the algorithm of Theorem 1.3 is optimal, up to constant additive terms in the exponent.

\textbf{Theorem 1.4.} There exists a constant \(c^* \in \mathbb{N}\) such that the following holds: Assuming #SETH, there is no integer \(k \geq 1\) such that #PERFMATCH can be solved in time \(O(n^{k-c^*})\) on n-vertex graphs G that are given together with a k-expression.

Thus in particular #PERFMATCH is not fixed-parameter tractable (assuming #SETH), and this holds also for the more general problem of computing the matching polynomial, answering the question of Makowsky et al. [73]. Our proof also shows that the problem is \(W[2]-hard.\)
Parameterizing by crossing number or genus. Finally, we turn our attention to graphs that are almost planar. A celebrated result of Fisher, Kasteleyn, and Temperley [63, 64, 82] showed that #PERF MATCH is polynomial-time solvable on planar graphs. If a graph is not planar, but a drawing with $k$ crossings is given, then a simple branching algorithm can reduce the problem to $2^k$ planar instances of #PERF MATCH.1

**Theorem 1.5.** Given an $n$-vertex planar graph $G$ with a drawing in the plane with $k$ crossings, #PERF MATCH can be solved in time $2^k n^{O(1)}$.

Thus the problem is polynomial-time solvable (and actually fixed-parameter tractable) for graphs of bounded crossing number; we avoid the discussion of how to find the drawing. More generally, Gallucio and Loebl [55], Tesler [83], and Regge and Zecchina [80] showed that the problem is fixed-parameter tractable even for graphs of bounded genus.

**Theorem 1.6.** ([55, 80, 83]) Given an $n$-vertex graph $G$ embedded on a surface of genus $g$, the problem #PERF MATCH can be solved in time $4^g n^{O(1)}$.

The base of the exponential is worse in Theorem 1.6 than in Theorem 1.5, which raises the obvious question of what the best possible base could be. While we cannot answer this question as tightly as in Theorems 1.2 and 1.4, we can at least show that the dependence on the parameter has to be exponential.

**Theorem 1.7.** Assuming #ETH, there is no $2^{o(k)} n^{O(1)}$ time algorithm for #PERF MATCH when given as input an $n$-vertex graph $G$ and a drawing of $G$ in the plane with $k$ crossings.

As the crossing number of a graph is always an upper bound on its genus, this theorem also gives a lower bound for the problem parameterized by genus.

Somewhat interestingly, this opposes a certain “square root phenomenon” that has been observed in the context of parameterized algorithms on planar graphs: for many decision problems, the best possible running times (assuming ETH) are often of the form $2^{O(\sqrt{k})} n^{O(1)}$ or $n^{O(\sqrt{k})}$ [24, 38–46, 52, 54, 65, 66, 77, 78, 84]. Thus one could have expected that the running time has a dependence of the form $2^{O(\sqrt{k})}$ on the number $k$ of crossings introduced into the planar drawing. However, Theorem 1.7 shows that this is not the case: the dependence has to be single-exponential. Similar violations of the square root phenomenon for counting problems have also been observed for #PERF MATCH on $k$-apex graphs, where a $n^{\Omega(k/\log k)}$ lower bound is known [32].

**Reductions among counting problems.** If the decision version of a problem is polynomial-time solvable, then hardness of the counting version cannot be proved by a parsimonious reduction from an NP-hard problem, as this would imply that the decision version of the NP-hard problem is polynomial-time solvable. Therefore, such hardness results necessarily involve a “mysterious” step that is highly non-parsimonious and usually very specific to the particular reduction source and target.

A standard way of doing this for #PERF MATCH introduces a weighted version of the problem: each edge is equipped with a weight, the weight of a perfect matching is defined as the product of the weights of its edges, and the task is to compute the sum of the weights of all the perfect matchings. Crucially, weights can be negative, thus allowing different perfect matchings to cancel out each other. After proving hardness for the weighted version, the negative weights can be eliminated by modular arithmetic [85] or polynomial interpolation [7, 30, 37], yielding hardness of the unweighted problem under Turing reductions. It is precisely at this step that the reduction becomes non-parsimonious: the existence of a perfect matching in one of the constructed instances does not tell directly whether the source instance has a solution or not.

While this technique is very powerful, we do not want to repeat it all over again every time a lower bound is proved for some counting problem. Therefore, we hide these steps behind a layer of abstraction, the so-called Holant framework: We first prove hardness for a certain type of Holant problem and then reduce this to #PERF MATCH in a second step. This last step encapsulates the arguments involving negative weights and polynomial interpolation, which allows us focus on our main task: constructing instances with bounded treewidth or cliquewidth. While the Holant framework introduces some notational overhead, the proofs become relatively transparent and comparable to similar lower bounds for decision problems.

**The Holant framework.** Let us briefly present the Holant framework (see Section 3 for more details), which was introduced by Valiant in the context of holographic algorithms [86], and which was extended into a more general framework since then [13, 15–20, 59, 60, 69].

A **signature graph** is a graph $\Omega$ with a weight $w(e)$ on each edge and a function $f_v : \{0, 1\}^{I(v)} \rightarrow \mathbb{Q}$ associated with each vertex $v$, where $I(v)$ is the set of edges incident to $v$. That is, $f_v$ assigns a rational value to every subset of edges of $I(v)$; we call $f_v$ the signature of $v$. Then for every subset $x \subseteq \{0, 1\}^{E(\Omega)}$ of edges, the functions $f_v$ determine a rational value at each vertex $v$ in a natural way. We define the value of $x$ to be the product of these values for
all $v \in V(\Omega)$ and we define the weight of a subset of edges to be the product of the weights of the edges. The Holant of $\Omega$ sums up, for every subset $x$ of edges, the product of the weight of $x$ and the value of $x$:

$$\text{Holant}(\Omega) = \sum_{x \subseteq \{(0,1)^{E(\Omega)} \mid e \in x\}} \prod_{v \in V(\Omega)} w(e) \prod_{v \in V(\Omega)} f_v(x_{I(v)}).$$

To approach this somewhat abstract problem, it may help to observe that it contains counting perfect matchings as a special case. Let $G$ be a graph, define $\Omega := G$, and assign $w(e) = 1$ to every $e \in E(\Omega)$. For every $v \in V(\Omega)$, define $f_v$ to be 1 if exactly one edge of $I(v)$ is selected and 0 otherwise. In this case, we also say that $f_v$ has the signature $\mathbb{W}_{\Omega}$, which is short for “Hamming weight $= 1$”. Then Holant$(\Omega)$ counts exactly those edge subsets $x \subseteq E(G)$ where $f_v(x) \neq 0$ at every vertex $v$, that is, where every vertex has degree exactly one in $x$. Therefore, one can think of Holant problems as a certain generalization of counting weighted perfect matchings and other degree-bounded subgraph counting problems.

In our proofs of lower bounds based on the Holant framework, we first reduce the problem of counting the satisfying assignments of a CNF-SAT formula to that of computing the Holant of a certain signature graph $\Omega$. Then we transform $\Omega$ to a signature graph $\Omega'$ in which every vertex has signature $\mathbb{W}_{\Omega'}$. This can equivalently be viewed as a weighted version of #PERF MATCH. Finally, a self-contained weight removal step shows that Holant$(\Omega')$ can be reduced to unweighted #PERF MATCH. The transformation from $\Omega$ to $\Omega'$ is performed using certain gadgets: if a vertex $v$ of degree $d$ features some signature $f_v$, then we replace it with a signature graph that has $d$ external edges and (in a well-defined sense) computes exactly the function $f_v$. The Holant framework provides a natural language for a rigorous treatment of this operation.

For our purposes, gadgets using only the signature $\mathbb{W}_{\Omega}$ are of particular interest; such gadgets are known as matchgates in the literature [14, 87]. It should be noted here that the established meaning of the term “matchgate” refers to planar gadgets, whereas in this paper, we deviate from this convention by explicitly allowing non-planar matchgates as well. This allows us to make the crucial observation that every signature of constant size (satisfying a trivially necessary parity condition) can be realized by a matchgate of constant size. Therefore, for our purposes, it is sufficient to construct a signature graph $\Omega$ where every vertex has the signature $\mathbb{W}_{\Omega}$ or has bounded degree. Then the rest of the reduction from Holant$(\Omega)$ to #PERF MATCH is completely automatic, we only need to verify that the these steps change treewidth or cliquewidth of the graph in a controlled way (but usually this is easy). We encapsulate this reduction from Holant problems to #PERF MATCH in Theorem 4.1. The hardness proofs in Theorems 1.2, 1.4, and 1.7 all construct Holant instances and then invoke Theorem 4.1 to complete the reduction to #PERF MATCH. We believe that this way of approaching hardness results for counting problems would also be fruitful for other problems and parameters.

## 2 Preliminaries

### 2.1 Complexity assumptions

The Exponential Time Hypothesis (ETH) conjectured by Impagliazzo, Paturi and Zane [62] and its strong variant (SETH) are conjectures about the exponential time complexity of $k$-SAT. Let $s_k$ be the infimum over all $\delta$ such that $n$-variable $k$-SAT can be solved in $2^{\delta n}$ time. Then ETH states that $s_3 > 0$. A simple and perhaps more intuitive consequence of ETH is that there is no $2^{o(n)}$ time algorithm for $n$-variable 3SAT, that is, no algorithm for 3SAT is subexponential in the number of variables. On its own, this may not rule out algorithms that are subexponential in the input size: the number of clauses can be superlinear in the number of variables. However, the Sparsification Lemma shows that ETH in fact rules out such algorithms as well. One way to formulate this result is the following:

**Lemma 2.1.** ([62]) Assuming ETH, $n$-variable $m$-clause 3SAT has no $2^{o(n+m)}$ time algorithm.

Since CNF-SAT with $n$ variables and $m$ clauses has a $2^n \cdot m^{O(1)}$ time algorithm, the sequence $\{s_k\}_{k \in \mathbb{N}}$ is bounded from above by $1$. Impagliazzo and Paturi [61] conjecture that 1 is indeed the limit of this sequence, a statement that became later known as SETH, the Strong Exponential Time Hypothesis [23]. We can formulate a convenient consequence of SETH by saying that $n$-variable $m$-clause CNF-SAT has no $(2 - \epsilon)^n m^{O(1)}$ time algorithm for any $\epsilon > 0$.

In the context of counting problems, it is natural to use the counting versions of ETH and SETH, which are defined analogously in terms of the counting problems #3SAT and #CNF-SAT (see [37]). Clearly, #ETH and #SETH are weaker assumptions than ETH and SETH, respectively, as they only rule out the existence of improved counting algorithms. Thus stating negative results assuming #ETH instead of ETH makes the result a priori stronger, but perhaps more importantly, it seems very natural and closer to the spirit of the problem to start our reductions from genuine counting problems.

Dell et. al [37] showed that the Sparsification Lemma can be made to work also for the counting version. Thus we have the following counting analog of Lemma 2.1.

**Lemma 2.2.** ([62]) Assuming #ETH, the problem #3SAT on formulas with $n$ variables and $m$ clauses cannot be solved in time $2^{o(n+m)}$.

Analogously to SETH, assuming #SETH has the consequence that $n$-variable $m$-clause #CNF-SAT has no $(2 - \epsilon)^n m^{O(1)}$ time algorithm for any $\epsilon > 0$. 

1653 

Unauthorized reproduction of this article is prohibited.
2.2 Inserting gadgets into graphs. Our graph notation is standard, but we need to introduce formal definitions for replacing vertices with gadgets in graphs. For this purpose, we find it convenient to use the notion of dangling edges, which are edges having only one endpoint [22].

Let $H_v$ be a graph with $d$ distinguished dangling edges $e_1, \ldots, e_d$ incident to distinct portal vertices $v_1, \ldots, v_d$, respectively. Let $G$ be a graph, let $v \in V(G)$ be of degree $d$, and assume that an ordering $f_1, \ldots, f_d$ of the $d$ edges incident to $v$ is given. Then the operation of inserting $H_v$ at $v$ is defined as follows: take the disjoint union of $G$ and $H_v$, remove $v$, and for every $1 \leq i \leq d$, remove $e_i$ and replace it with an edge connecting $v_i$ with the other endpoint of $f_i$.

We can extend this operation to inserting more than one graph in parallel. That is, assume we are given vertices $v^1, \ldots, v^t$ in $G$ (each vertex having a fixed ordering of the edges incident to it) and graphs $H_{v^1}, \ldots, H_{v^t}$ such that the number of dangling edges of $H_{v^i}$ and the degree of $v^i$ agree, we can define the insertion of $H_{v^i}$ at $v^i$ in parallel for $1 \leq i \leq t$ with the obvious meaning depicted in Figure 1.

2.3 Treewidth, pathwidth and cutwidth. We recall the most important notions related to treewidth in this section. A tree decomposition of a graph $G$ is a pair $(T, B)$ in which $T$ is a tree and $B = \{B_t | t \in V(T)\}$ is a family of subsets of $V(G)$ such that

1. $\bigcup_{t \in V(T)} B_t = V$;
2. for each edge $e = uv \in E(G)$, there exists a node $t \in V(T)$ such that $u, v \in B_t$, and
3. the set of nodes $\{t \in V(T) | v \in B_t\}$ forms a connected subtree of $T$ for every $v \in V(G)$.

To distinguish between vertices of the original graph $G$ and vertices of $T$ in the tree decomposition, we call vertices $i$ of $T$ nodes and their corresponding $B_i$'s bags. The width of the tree decomposition is the maximum size of a bag in $B$ minus 1. The treewidth of a graph $G$, denoted by $tw(G)$, is the minimum width over all possible tree decompositions of $G$.

A path decomposition is a tree decomposition where $T$ is a path. The pathwidth $pw(G)$ of a graph $G$ is the minimum width over all possible path decompositions of $G$. By definition, we have $tw(G) \leq pw(G)$.

A linear layout of an $n$-vertex graph $G$ is an ordering $v_1, \ldots, v_n$ of $V(G)$. For $1 \leq i \leq n$, the cut after $v_i$ is the set of edges between $\{v_1, \ldots, v_i\}$ and $\{v_{i+1}, \ldots, v_n\}$. The cutwidth of the linear layout is the maximum size of the cut after $v_i$ for $1 \leq i < n$. The cutwidth of $G$, denoted by $cutw(G)$, is the minimum cutwidth over all possible linear layouts of $G$. A linear layout of cutwidth $cutw(G)$ is called optimal. It is easy to see that $pw(G) \leq cutw(G)$.

The following lemma shows that inserting gadgets of bounded cutwidth increases cutwidth only by a constant (this property of cutwidth is the main reason we are using it in our proofs).

**Lemma 2.3.** Let $G$ be a graph and let $X \subseteq V(G)$ be a subset of vertices, each of degree at most $d$. For every $v \in X$, let us replace $v$ by inserting a graph $H_v$ of cutwidth at most $c$; let $G'$ be the resulting graph. Then $cutw(G') = cutw(G) + d + c$.

2.4 Cliquewidth. We follow Fomin et al. [51] for the basic definitions of cliquewidth. Let $G$ be a graph and $t$ be a positive integer. A $t$-graph is a graph with vertices labeled by integers from $\{1, 2, \ldots, t\}$. We refer to a $t$-graph consisting of exactly one vertex labeled by some $i \in \{1, 2, \ldots, t\}$ as an initial $t$-graph. The cliquewidth $cw(G)$ is the smallest integer $t$ such that $G$ can be constructed by repeated applications of the following four operations: $i(v)$: Introduce operation constructing an initial $t$-graph with vertex $v$ labeled by $i$, $+:$ Disjoint union of two $t$-graphs, $\rho_i \rightarrow j$: Relabel operation changing all labels $i$ to $j$, and $\eta_i \rightarrow j$: Join operation making all vertices labeled by $i$ adjacent to all vertices labeled by $j$.

An expression tree of a graph $G$ is a rooted tree $T$ with nodes of four types $i$, $\oplus$, $\rho$, and $\eta$, corresponding to the operations described above. To each node, a $t$-graph is associated such that $G$ is isomorphic to the graph corresponding to the root of $T$ after removal of all labels.

**Introduce nodes** $i(v)$ are precisely the leaves of $T$, and each such node corresponds to an initial $t$-graph consisting of the $i$-labeled vertex $v$.

**Union node** $\oplus$ stands for a disjoint union of graphs associated with its children.

**Relabel node** $\rho_i \rightarrow j$ has one child and is associated with the $t$-graph obtained by the application of the relabeling operation to the graph corresponding to its child.

**Join node** $\eta_i \rightarrow j$ has one child and is associated with the $t$-graph resulting from the application of the join operation to the graph corresponding to its child.

The width of the expression tree $T$ is the number of different labels appearing in $T$. If $G$ is of cliquewidth $t$, then there is a rooted expression tree $T$ of width $t$ for $G$. An expression tree $T$ is irredundant if for any join node $\eta_i \rightarrow j$, the vertices labeled by $i$ and $j$ are not adjacent in the graph associated with its child. It was shown by Courcelle and Olariu [29] that every expression tree $T$ of $G$ can be transformed into an irredundant expression tree $T'$ of the same width in time linear in the size of $T$. We will use this for algorithmic purposes.

The following definitions related to clique expressions are nonstandard, but we find them very useful for
For every vertex $v$ in $G$, let $G$ be a graph with edge weights $w$ on each edge $e$.

Let $G_v$ be the subexpression associated with $v$. We prove that inserting bounded-pathwidth gadgets, we can verify that the proof provides a $\rho$-expression featuring only singular vertices.

We say that an edge $uv$ is singular in an irredundant clique expression if the unique operation $\eta_{v,j}$ creating $uv$ created exactly one edge (i.e., there was exactly one vertex with label $i$ and exactly one vertex with label $j$ at this point). We say that label $i$ is a forget label if there is no operation $i(v)$, $\rho_{v,j}$, or $\eta_{i,j}$ in the $k$-expression for any $j$ (thus label $i$ may appear only in an operation of the form $\rho_{j-i}$). We say that a vertex $v$ is singular if every edge incident to $v$ is singular and whenever the graph associated to a subexpression contains $v$, then $v$ has either a singleton label or a forget label.

It is known and easy to see that $\text{cw}(G) \leq \text{pw}(G) + 2$ [49] and we can verify that the proof provides a $(1, \text{pw}(G) + 1)$-expression featuring only singular vertices. We prove that inserting bounded-pathwidth gadgets at singular vertices increases the cliquewidth only moderately: only the number of singleton labels is increased.

**Lemma 2.4.** Let $G$ be a graph and let $X \subseteq V(G)$ be a subset of vertices, each of degree at most $d$. Suppose that $G$ has a $(k, s)$-expression $T$ where every $v \in X$ is singular. For every $v \in X$, replace $v$ by inserting a graph $G_v$ of pathwidth at most $p$; let $G'$ be the resulting graph. Then $\text{cw}(G') \leq k + d(s + 1) + p + 1$.

**3 Holants and Matchgates**

We give an introduction to what we call the Holant framework, a toolbox based on [21, 22, 87]. A more detailed exposition can be found in Chapter 2 of [31].

**3.1 Holants.**

Given a graph $G$ and $v \in V(G)$, we denote the edges incident with $v$ by $I(v)$.

**Definition 1.** A signature graph is an edge-weighted graph $\Omega$, which may feature parallel edges, and which has a signature $f_v : \{0, 1\}^{I(v)} \to \mathbb{Q}$ associated with each vertex $v \in V(\Omega)$.

The Holant of $\Omega$ is a particular sum over the edge assignments $x \in \{0, 1\}^{E(\Omega)}$. Given an assignment $x \in \{0, 1\}^{E(\Omega)}$, we say that an edge $e \in E(\Omega)$ is active in $x$ if $x(e) = 1$ holds, otherwise $e$ is inactive in $x$. We tacitly identify $x$ with the set of active edges in $x$. Given a subset $S \subseteq E(\Omega)$, we write $x|_S$ for the restriction of $x$ to $S$, which is the unique assignment in $\{0, 1\}^S$ that agrees with $x$ on $S$.

**Definition 2.** ([87]) Let $\Omega$ be a signature graph with edge weights $w : E(\Omega) \to \mathbb{Q}$ and a function $f_v : \{0, 1\}^{I(v)} \to \mathbb{Q}$ for each $v \in V(\Omega)$. Furthermore, let $x \in \{0, 1\}^{E(\Omega)}$ be an assignment to the edges of $\Omega$. Then we define

\[
\text{val}_\Omega(x) := \prod_{v \in V(\Omega)} f_v(x|_{I(v)}),
\]

\[
\text{w}_\Omega(x) := \prod_{e \in x} w(e),
\]

and we say that $x$ satisfies $\Omega$ if $\text{val}_\Omega(x) \neq 0$ holds. Furthermore, we define

\[
\text{Holant}(\Omega) := \sum_{x \in \{0, 1\}^{E(\Omega)}} \text{w}_\Omega(x) \cdot \text{val}_\Omega(x).
\]

A particularly useful type of signatures is that of Boolean functions, whose ranges are restricted to $\{0, 1\}$ rather than $\mathbb{Q}$. If all signatures appearing in $\Omega$ are Boolean, then Holant($\Omega$) simply sums over those assignments $x \in \{0, 1\}^{E(\Omega)}$ that pass all constraints imposed by the vertex functions, and each $x$ is weighted by $\text{w}_\Omega(x)$. We use the following Boolean signatures:
3.2 Gates and matchgates. Given a signature graph $\Omega$, we can sometimes simulate signatures by gadgets or gates, which are signature graphs with dangling edges. A dangling edge is an “edge” with only one endpoint. These notions are borrowed from the $F$-gates in [22].

DEFINITION 4. For disjoint sets $A$ and $B$, and for assignments $x \in \{0,1\}^A$ and $y \in \{0,1\}^B$, we write $xy \in \{0,1\}^{A \cup B}$ for the assignment that agrees with $x$ on $A$, and with $y$ on $B$. We also say that the assignment $xy$ extends $x$.

A gate is a signature graph $\Gamma$, possibly containing a set $D \subseteq E(\Gamma)$ of dangling edges, all of which have edge weight 1. A gate $\Gamma$ is a matchgate if it features only the signature $\mathbb{H}_w = 1$. The signature realized by $\Gamma$ is the function $\text{Sig}(\Gamma) : \{0,1\}^D \rightarrow \mathbb{Q}$ that maps $x$ to

$$\text{Sig}(\Gamma, x) = \sum_{xy \in \{0,1\}^{E(\Gamma)}} w_\Gamma(xy) \cdot \text{val}_\Gamma(xy).$$

We consider the dangling edges $D$ of gates $\Gamma$ to be labeled as $1, \ldots, |D|$. This way, we can consider signatures $\text{Sig}(\Gamma)$ as functions of type $\{0,1\}^{|D|} \rightarrow \mathbb{Q}$ instead of $\{0,1\}^D \rightarrow \mathbb{Q}$.

REMARK 1. Note that we allow matchgates to be non-planar. This deviates from the notion of matchgates established in the literature [14, 87], which are required to be planar. More precisely, a matchgate on dangling edges 1, \ldots, $d$ is planar if it can be drawn in the plane without crossings such that its dangling edges appear in the order 1, \ldots, $d$ on its outer face.

The usefulness of gates for our arguments lies in their ability of simulating complex signatures by simpler signatures. For instance, for any even arity $k \in \mathbb{N}$, we can realize the $k$-ary signature

$$\text{EQ}_k(x_1, \ldots, x_k) := [x_1 = \ldots = x_k]$$

by a gate whose vertices feature only the equality signature $\text{EQ}_4$ of arity 4.

EXAMPLE 1. For all even $k \geq 4$, there exists a gate $\Gamma_{\text{EQ}}$ with $\text{Sig}(\Gamma_{\text{EQ}}) = \text{EQ}_k$. This gate consists of vertices $v_1, \ldots, v_{k-1}$, each equipped with $\text{EQ}_4$, internal edges $e_1, \ldots, e_{k-2}$ with $e_i = v_i v_{i+1}$ for all $i$, and $k$ additional dangling edges, as shown in Figure 2.

![Figure 2](image.png)

Figure 2: The gate $\text{EQ}_1$ with $\text{Sig}(\Gamma_{\text{EQ}}) = \text{EQ}_1$ in Example 1.

Its correctness can be verified from the definition: an assignment to the dangling edges of $\Gamma_{\text{EQ}}$ can be extended to a satisfying assignment for $\Gamma_{\text{EQ}}$ only if all values agree, in which case there is exactly one assignment.

In the following, we formalize the operation of inserting a gate $\Gamma$ into a signature graph so as to simulate a vertex of a specific signature.

DEFINITION 5. Let $\Omega$ be a signature graph, let $v \in V(\Omega)$ with $D = I(v)$ and let $\Gamma$ be a gate with dangling edges $D$.

Then we can insert $\Gamma$ at $v$ by (i) deleting $v$ and keeping $D$ as dangling edges, and then (ii) placing a copy of $\Gamma$ into $\Omega$ and identifying each dangling edge $e \in D$ across $\Gamma$ and $\Omega$. That is, if $e$ has an endpoint $u$ in $\Omega$, and an endpoint $v$ in $\Gamma$, then we consider $e$ as an edge $uv$ in the resulting graph.

A simple calculation shows that insertions of suitable matchgates preserve Holants.

LEMMA 3.1. Let $\Omega$ be a signature graph and let $v \in V(\Omega)$. Let $\Omega'$ be derived from $\Omega$ by inserting a gate $\Gamma$ with $\text{Sig}(\Gamma) = f$, at $v$. Then $\text{Holant}(\Omega) = \text{Holant}(\Omega')$.

We focus on matchgates, as defined above, since these allow us to reduce the problem of computing Holants to #PERFMATCH. Note that only graphs with an even number of vertices admit perfect matchings. This implies the following parity condition on signatures of matchgates.

FACT 3.1. (Parity Condition) If a signature $f : \{0,1\}^d \rightarrow \mathbb{Q}$ can be realized by a matchgate, then at least one of the following holds:

- For all $x \in \{0,1\}^d$ with odd $\text{hw}(x)$, we have $f(x) = 0$. Then we call $f$ even.
- For all $x \in \{0,1\}^d$ with even $\text{hw}(x)$, we have $f(x) = 0$. Then we call $f$ odd.

We say that $f$ is parity-consistent if it is even or odd. We also say that a signature graph is parity-consistent if every vertex has a parity-consistent signature.

Furthermore, we can summarize the use of matchgates in the following fact. To this end, given a graph $G$ with edge weights $w : E(G) \rightarrow \mathbb{Q}$, let us define

$$\text{PerfMatch}(G) = \sum_{M \subseteq E(G)} \prod_{e \in M} w(e),$$

where $M$ ranges over the perfect matchings $M \subseteq E(G)$ of $G$. This was also defined in [87].
we obtain a graph $G$ black, respectively. A comparable matchgate is shown in
[72x416]bra system, in a process that is detailed in Appendix C of
[72x149]EQ
[72x161]k
[72x161]>[72x299]Γ
[72x543]Edges of weight
[72x464]Γ
[72x231]realizing
[72x231]EQ
[72x311]properties. In the following, we use this to realize the signature
[72x335]natures to construct matchgates for more complex signa-
[72x358]As
[72x358]3.3 Matchgates for parity-consistent signatures.
[72x403]∑
[72x403]v as a gate at $v$, as specified in
[72x462]FACT 3.2.
[79x402]LEMMA 3.3.
[79x402]LEMMA 3.2.
[82x309]N
[82x309]∈
[82x172]=
[161x173]4.F o r
[161x416]G
[161x416]∑
[161x416]v
[161x414], there is a matchgate
[161x414]Ω
[161x414]on
[161x415](
[161x415]V
[161x415])
[161x416]vertices and
[161x417]∑
[161x417]v
[161x414]|E(Γv)| edges with

Holant($Ω$) = PerfMatch($G(Ω)$).

3.3 Matchgates for parity-consistent signatures. As
seen in Example 1, we can use matchgates for simple sig-
natures to construct matchgates for more complex signa-
tures. In the following, we use this to realize the signature
$EQ_k$ for even $k \in \mathbb{N}$ from the matchgate $Γ_k$ for $EQ_4$ shown in
Figure 3. We discovered $Γ_k$ using a computer algebra system, in a process that is detailed in Appendix C of
[31]. Note that $EQ_k$ for odd $k \in \mathbb{N}$ does not satisfy the par-
ity condition and hence cannot be realized by matchgates.

LEMMA 3.2. For all even $k \in \mathbb{N}$, there is a matchgate realizing $EQ_k$. This matchgate features $O(k)$ vertices and edges and only $-1, \frac{1}{2}, 1$ as edge weights.

Proof. For $k = 2$, we use a path on 4 vertices with
dangling edges at its endpoints. For $k = 4$, we can verify that the matchgate $Γ_4$ in Figure 3 realizes $EQ_4$. For
$k > 4$, we use the gate from Example 1 and realize each occurrence of $EQ_4$ by $Γ_k$.

The parity condition in Fact 3.1 tells us that every
signature of a matchgate is even or odd. In the following,
we show that the converse holds as well:

LEMMA 3.3. Let $f : \{0,1\}^d \rightarrow \Lambda$ be a signature of arity
$d \in \mathbb{N}$ that is parity-consistent. Then there is a matchgate

$Γ$ that realizes $f$. Furthermore, if $supp(f)$ denotes the
support of $f$, then

- $Γ$ has $O(|supp(f)| \cdot d)$ vertices and edges,
- $Γ$ has maximum degree at most $|supp(f)| + O(1)$,
- the edge weights of $Γ$ are contained in the set $I \cup \{−1,\frac{1}{2}, 1\}$,
- given as input $\{(x,f(x)) \mid x \in supp(f)\}$, we can construct $Γ$ in time $O(|supp(f)| \cdot d)$.

The construction of $Γ$ resembles the construction of a
formula in DNF from a given Boolean function. For each
element $x \in supp(f)$, we create an assignment gate $L_x$ that
tests whether the dangling edges of $Γ$ are assigned $x$ and
this way ensures that $Sig(Γ,x) = f(x)$. Furthermore, we ensure that $Sig(Γ,x) \neq 0$ holds only if exactly one of these
tests succeeds.

Proof. of Lemma 3.3. For this proof, we assume that
d − $hw(x)$ is odd for every $x \in supp(f)$, which implies
that exactly one of $d$ and $f$ is even, while the other one is odd. If $d − hw(x)$ is even, the proof proceeds similarly.

We first define the following gate $Γ'$ on dangling edges $[d]$, shown exemplarily in Figure 4. The matchgate
$Γ$ is then obtained by realizing all signatures appearing in
$Γ'$ by matchgates.

1. Create vertices $O = \{o_1,\ldots,o_d\}$ with signature $HW=1$, and for $i \in [d]$, add the dangling edge $i$ and
make it incident to $o_i$.
2. Create a vertex $a$ with signature $HW=1$.
3. Let $supp(f) = \{x_1,\ldots,x_r\}$ for some $r \in \mathbb{N}$. For each
$κ \in [r]$, let $S_κ = O \backslash \{o_i \mid i \in x_κ\}$. Note that $|S_κ|$ is
odd by assumption. We perform the following steps.

(a) Create a vertex $v_κ$ with signature $EQ_{|S_κ|+1}$ and
make it adjacent to all vertices in $S_κ$. Note that
We prove that $\Gamma'$ realizes $f$. Let $y \in \{0,1\}^{E(\Gamma')}$ be a satisfying assignment. By $\text{HW}_{x=1}$ at the vertex $a$ and $\text{EQ}$ at $v_\kappa$ for $\kappa \in [r]$, there is exactly one $\kappa \in [r]$ such that all edges of $I(v_\kappa)$ are active under $y$, while all edges in $I(v_{\kappa'})$ for $\kappa' \neq \kappa$ are inactive. In particular, we then have

$$\text{val}_{\Gamma'}(y) \cdot w_{\Gamma'}(y) = f(x_\kappa).$$

Let $x = y|_d$ be the restriction of $y$ to the dangling edges of $\Gamma'$. We observe that, if the edges in $I(v_\kappa)$ are active under $y$, for $\kappa \in [r]$, then $x = x_\kappa$; Since $y$ is satisfying, by $\text{HW}_{x=1} at O$, every $o_i \in O$ for $i \in [d]$ is incident with exactly one active edge, and this edge must be dangling if $x(i) = 1$, or contained in $I(v_\kappa)$ if $x(i) = 0$. Hence, for every $x \in \{0,1\}^d$, there is a satisfying assignment $y$ of $\Gamma'$ that extends $x$ if and only if $x = x_\kappa$ for some $\kappa \in [s]$. Furthermore, in this case, $y$ is unique and satisfies (3.5). We conclude that $\text{Sig}(\Gamma', x) = f(x)$.

4 From Holants to perfect matchings

In this section, we present our main technical tool that encapsulates the reduction from Holant problems to unweighted $\text{PERF MATCH}$. From the previous section, we know that bounded-arity signatures can be replaced with matchgates that may feature edge weights (which are possibly even negative). In the following, we show how to reduce the computation of Holant($\Omega$) for $\kappa \in [r]$ to unweighted $\text{#PERF MATCH}$, it remains to remove the edge weights $-1$ and $1/2$ from $G$. To this end, we apply a standard interpolation argument: Introduce two indeterminates $x, y$ and replace each edge weight $-1$ by $x$, and each edge weight $1/2$ by $y$. This yields a graph $G_{x,y}$ on edge weights $1, x, y$. Then

$$p(x,y) := \text{PerfMatch}(G_{x,y}) \in \mathbb{Z}[x,y]$$

is a polynomial of maximum degree $\ell := |V(G)|/2$ in the indeterminates $x$ and $y$. Assume we can evaluate $p(\xi)$ on all points $\xi \in A^2$ for an arbitrary set $A \subseteq \mathbb{Q}$ with $|A| = \ell + 1$. That is, we can evaluate $\text{PerfMatch}(G_{\xi})$ for graphs on edge weights $\{1\} \cup A$. Then we can use multivariate polynomial interpolation (see [30]) to obtain all coefficients of $p$ in time $O(1)$. In particular, we can then evaluate $p(-1,1/2) = \text{PerfMatch}(G)$.

In the following, the set $A$ will be chosen according to the parameter we wish to bound. We then evaluate $p(\xi)$ for $\xi \in A^2$ by means of certain gadgets/matchgates that simulate the edge weights from $A$. To this end, we present two different ways of choosing $A$ and simulating the edge weights (see Figure 5).

Method 1. We let $A = \{2^i \mid i \in [\ell]\}$ and observe that an edge $ab$ of weight $2^i$ can be simulated with a path of length $2i - 1$ on edges $e_1, \ldots, e_{2i-1}$, where edge $e_{2^i-1}$ for $\kappa \in [i]$ has weight $2$. Then an edge of weight $2$ between vertices $u$ and $v$ can in turn be simulated by two parallel edges between $u$ and $v$, each of which is subdivided twice. We obtain a gadget of cutwidth $2$ on $O(\ell)$ vertices.

Method 2. We let $A = [\ell]$ and use a construction from [37] to simulate the weight $i \in [\ell]$ with $O(\log^2 i)$ vertices and path width $2i$. It is clear that we can simulate an edge of weight $r + s$ by two parallel edges of weights $r$ and $s$, respectively. Hence, we can simulate weight $i \in [\ell]$ by at most $|\log i|$ parallel edges by introducing an edge for each 1-bit in the binary representation of $i$: if the $\kappa$-th bit is 1, then we introduce a parallel edge of weight $2^\kappa - 1$. Then we realize each weight $2^\kappa - 1$ by a path on $O(\kappa)$ vertices as in Method 1.

Using interpolation as described above, we can then reduce the computation of Holant($\Omega$) to unweighted $\text{PERF MATCH}$ with either of the two methods. To ensure that one of the three chosen statements holds, we choose between the two methods in the following way:

\[ |S_\kappa| + 1 \text{ is even, so } \text{HW}_x = 1 \text{ can be realized by a matchgate by Lemma 3.3.} \]

\[ (b) \text{ Draw an edge of weight } f(x_\kappa) \text{ from } v_\kappa \text{ to } a. \]

\[ \text{Proof. By Lemma 3.3, we obtain a matchgate } \Gamma_v \text{ on } O(2^d) \text{ vertices for each parity-consistent signature } f_v \text{ at } v \in V(\Omega) \text{ of arity } d. \text{ Let } G \text{ denote the graph on edge weights } \{-1,1/2\} \text{ obtained from } \Omega \text{ by inserting } \Gamma_v \text{ at } v, \text{ for all } v. \text{ By Fact 3.2, we have Holant(}\Omega) = \text{PerfMatch}(G). \]

To reduce to unweighted $\text{#PERF MATCH}$, it remains to remove the edge weights $-1$ and $1/2$ from $G$. To this end, we apply a standard interpolation argument: Introduce two indeterminates $x, y$ and replace each edge weight $-1$ by $x$, and each edge weight $1/2$ by $y$. This yields a graph $G_{x,y}$ on edge weights $1, x, y$. Then

$$p(x,y) := \text{PerfMatch}(G_{x,y}) \in \mathbb{Z}[x,y]$$

is a polynomial of maximum degree $\ell := |V(G)|/2$ in the indeterminates $x$ and $y$. Assume we can evaluate $p(\xi)$ on all points $\xi \in A^2$ for an arbitrary set $A \subseteq \mathbb{Q}$ with $|A| = \ell + 1$. That is, we can evaluate $\text{PerfMatch}(G_{\xi})$ for graphs on edge weights $\{1\} \cup A$. Then we can use multivariate polynomial interpolation (see [30]) to obtain all coefficients of $p$ in time $O(1)$. In particular, we can then evaluate $p(-1,1/2) = \text{PerfMatch}(G)$.

In the following, the set $A$ will be chosen according to the parameter we wish to bound. We then evaluate $p(\xi)$ for $\xi \in A^2$ by means of certain gadgets/matchgates that simulate the edge weights from $A$. To this end, we present two different ways of choosing $A$ and simulating the edge weights (see Figure 5).

Method 1. We let $A = \{2^i \mid i \in [\ell]\}$ and observe that an edge $ab$ of weight $2^i$ can be simulated with a path of length $2i - 1$ on edges $e_1, \ldots, e_{2i-1}$, where edge $e_{2^i-1}$ for $\kappa \in [i]$ has weight $2$. Then an edge of weight $2$ between vertices $u$ and $v$ can in turn be simulated by two parallel edges between $u$ and $v$, each of which is subdivided twice. We obtain a gadget of cutwidth $2$ on $O(\ell)$ vertices.

Method 2. We let $A = [\ell]$ and use a construction from [37] to simulate the weight $i \in [\ell]$ with $O(\log^2 i)$ vertices and path width $2i$. It is clear that we can simulate an edge of weight $r + s$ by two parallel edges of weights $r$ and $s$, respectively. Hence, we can simulate weight $i \in [\ell]$ by at most $|\log i|$ parallel edges by introducing an edge for each 1-bit in the binary representation of $i$: if the $\kappa$-th bit is 1, then we introduce a parallel edge of weight $2^\kappa - 1$. Then we realize each weight $2^\kappa - 1$ by a path on $O(\kappa)$ vertices as in Method 1.

Using interpolation as described above, we can then reduce the computation of Holant($\Omega$) to unweighted $\text{PERF MATCH}$ with either of the two methods. To ensure that one of the three chosen statements holds, we choose between the two methods in the following way:

\[ |S_\kappa| + 1 \text{ is even, so } \text{HW}_x = 1 \text{ can be realized by a matchgate by Lemma 3.3.} \]

\[ (b) \text{ Draw an edge of weight } f(x_\kappa) \text{ from } v_\kappa \text{ to } a. \]
Figure 5: Removing weights using Method 1 and Method 2 in the proof of Theorem 4.1.

In the case of cutwidth, we use Method 1. Every weighted matchgate \( \Gamma \) to be inserted into \( \Omega \) has size bounded by a function of \( d \), hence its cutwidth is also bounded by a function of \( d \). Method 1 can be expressed as subdividing edges of \( \Gamma \) a certain number of times, replacing some edges with two parallel edges, and then again subdividing some edges. Subdividing edges does not increase cutwidth and duplicating edges increases cutwidth at most by a factor of 2, hence the matchgates have cutwidth bounded by a function of \( d \) even after simulating the weights. Then Lemma 2.3 shows that inserting these gadgets increases cutwidth at most by a function of \( d \).

In the case of crossing number, we use Method 1. We can extend the planar drawing of \( \Omega \) to a drawing of \( G \) that features crossings only in the drawings of nonplanar matchgates \( \Gamma \). Subdividing edges does not increase the crossing number, duplicating edges increases it at most by a factor of 4. Thus each nonplanar matchgate has crossing number bounded by some function of \( d \), even after simulation of weights. Since \( \Omega \) features at most \( (|X| - t) \) nonplanar matchgates, we obtain that \( G \) has crossing number at most \( (|X| - t) \cdot c_d \) for some suitably large constant \( c_d \). Note that no crossings are introduced into planar matchgates by simulating edge-weights.

In the case of cliquewidth, we use Method 2. Every weighted matchgate \( \Gamma \) has size bounded by a function of \( d \), hence its cliquewidth is also bounded by a function of \( d \). Method 2 can be expressed as replacing edges with several parallel copies, subdividing edges several times, duplicating certain edges, and then again subdividing them. Replacing an edge with parallel edges has no effect on cliquewidth and subdividing a subset of the edges (even multiple times) can increase pathwidth at most by 1, as shown in [5]. Thus, after the application of Method 2, the unweighted gadgets still have pathwidth bounded by a function of \( d \). Then by Lemma 2.4, the graph \( G' \) obtained after the insertion of the weighted gadgets has cliquewidth at most \( k + d(s + 1) \) plus a constant depending only on \( d \), hence we can indeed bound \( \text{cw}(G') \) by \( k + s \cdot c_d \) for a suitably large constant \( c_d \). This concludes the proof.

5 Parameterizing by cutwidth or crossing number

In this section, we prove Theorems 1.2 and 1.7 by a lower bound for parameterization by cutwidth (implying the lower bound for parameterization by pathwidth and treewidth) and by a lower bound for parameterization by crossing number (implying the lower bound for parameterization by genus).

We first show how to reduce counting the number of satisfying assignments of an \( n \)-variable \( m \)-clause \( d \)-CNF formula \( \varphi \) to computing the Holant of certain signature graphs. Then we invoke Theorem 4.1 to reduce the resulting Holant problem to \#PERF MATCH on graphs with \( O(nm) \) vertices and edges that have cutwidth \( n + O(1) \) or crossing number \( O(n + dm) \).

5.1 Parameterizing by cutwidth. Let \( \varphi \) be a \( d \)-CNF formula on \( n \) variables \( x_1, \ldots, x_n \) and \( m \) clauses \( C_1, \ldots, C_m \). As an intermediate step in the reduction to \#PERF MATCH, we derive two signature graphs from \( \varphi \), following the structure outlined in Figure 6.

That is, each graph consists essentially of an \( n \times m \) square grid whose rows and columns correspond to the variables and clauses of \( \varphi \), respectively. The horizontal
edges in row \(i \in [n]\) will serve to propagate a binary assignment to the variable \(x_i\), and in column \(j \in [m]\), we will test whether the assignment to \(x_1, \ldots, x_n\) encoded this way satisfies clause \(C_j\).

To realize this construction, we need to define certain auxiliary Boolean signatures that play the role of negative and positive (and neutral) literals in \(\phi\).

**Definition 6.** We define the Boolean 6-ary signatures \(\text{LIT}_0\), \(\text{LIT}_-\) and \(\text{LIT}_+\) on the following six inputs, grouped into three pairs of inputs

\[
(\text{x}_{\text{in}}, \text{x}_{\text{out}}), (\text{x}_{\text{top}1}, \text{x}_{\text{top}2}), (\text{x}_{\text{btm}1}, \text{x}_{\text{btm}2}).
\]

For any assignment \(a \in \{0, 1\}^6\), the signatures yield value 1 if all of the following holds: Firstly, for each input pair, the two values in the pair agree. That is, we have \(a(\text{x}_{\text{in}}) = a(\text{x}_{\text{out}})\) and \(a(\text{x}_{\text{top}1}) = a(\text{x}_{\text{top}2})\) and \(a(\text{x}_{\text{btm}1}) = a(\text{x}_{\text{btm}2})\).

Secondly, if the top input pair is active, then so is the bottom input pair, regardless of the assignment to any other inputs. That is, if \(a(\text{x}_{\text{top}1}) = a(\text{x}_{\text{top}2}) = 1\), then \(a(\text{x}_{\text{btm}1}) = a(\text{x}_{\text{btm}2}) = 1\).

On the other hand, if the top input pair is inactive, that is, \(a(\text{x}_{\text{top}1}) = a(\text{x}_{\text{top}2}) = 0\), then \(\text{LIT}_0\), \(\text{LIT}_-\) and \(\text{LIT}_+\) differ in their behavior:

- For \(\text{LIT}_0\), we require \(a(\text{x}_{\text{btm}1}) = a(\text{x}_{\text{btm}2}) = 0\).
- For \(\text{LIT}_-\), we require \(a(\text{x}_{\text{btm}1}) = a(\text{x}_{\text{btm}2}) = a(\text{x}_{\text{in}})\).
- For \(\text{LIT}_+\), we require \(a(\text{x}_{\text{btm}1}) = a(\text{x}_{\text{btm}2}) = a(\neg \text{x}_{\text{in}})\).

It is evident that \(\text{LIT}_0\), \(\text{LIT}_-\) and \(\text{LIT}_+\) are all even signatures. As a matter of fact, the inputs \(\text{x}_{\text{top}1}\), \(\text{x}_{\text{top}2}\) and \(\text{x}_{\text{btm}1}\), \(\text{x}_{\text{btm}2}\) are defined to come in pairs with enforced equality for no other reason than to ensure this.

All signatures defined above propagate the assignment of \(\text{x}_{\text{in}}\) to \(\text{x}_{\text{out}}\). This ensures that, in every satisfying assignment to the constructed signature graph, the same binary value will be assigned to all horizontal edges. If \(\text{x}_{\text{top}1} = 1\), then we require \(\text{x}_{\text{btm}1} = 1\) as well. This ensures that if a clause is satisfied by previous literals, then this information will be propagated to the next literal. If \(\text{x}_{\text{top}1} = 0\), then \(\text{LIT}_0\) simply passes this information to \(\text{x}_{\text{btm}1}\). The signatures \(\text{LIT}_-\) and \(\text{LIT}_+\) however check whether \(\text{x}_{\text{in}}\) is assigned 1 (in which case a positive literal would be satisfied and \(\text{LIT}_+\) assigns \(\text{x}_{\text{in}}\) to \(\text{x}_{\text{btm}1}\)) or 0 (in which case \(\text{LIT}_-\) assigns \(\neg \text{x}_{\text{in}}\) to \(\text{x}_{\text{btm}1}\)). We can now define the relevant signature graphs from \(\phi\).

**Lemma 5.1.** Let \(\phi\) be a \(d\)-CNF formula on \(n\) variables and \(m\) clauses. Then we can construct two unweighted signature graphs \(\Omega_0\) and \(\Omega_1\) such that

\[
\text{(5.6)} \quad \text{SAT}(\phi) = \text{Holant}(\Omega_0) + \text{Holant}(\Omega_1).
\]

Furthermore, both \(\Omega_0\) and \(\Omega_1\) are planar, have cutwidth \(n + O(1)\) and maximum degree \(O(1)\).

**Proof.** We define the graphs underlying \(\Omega_0\) and \(\Omega_1\) as follows, see also Figure 6:

1. Create an \(n \times m\) grid of vertices \(v_{i,j}\) for \(i \in [n]\) and \(j \in [m]\). Copy each vertical edge, such that \(v_{i,j}\) and \(v_{i,j+1}\) are connected by two parallel edges for \(i \in [n-1]\) and \(j \in [m]\). We will later describe how to assign signatures to these vertices.

2. Create two additional sets of vertices \(\{v_{i,0} \mid i \in [n]\}\) and \(\{v_{j,m+1} \mid i \in [n]\}\). For each \(i \leq n - 1\), connect \(v_{i,0}\) to \(v_{i+1,0}\) and to \(v_{i,1}\). Connect \(v_{n,0}\) to \(v_{n,1}\). Also connect \(v_{i,m+1}\) to \(v_{i+1,m+1}\) and to \(v_{i,m}\) for each \(i \leq n - 1\). Connect \(v_{n,m+1}\) to \(v_{n,m}\).

3. In the case of \(\Omega_0\), assign the signature \(\text{EVEN}\) of appropriate arity to each vertex created in the previous step. Define \(\Omega_1\) likewise, but assign \(\text{ODD}\) to the two vertices \(v_{n,0}\) and \(v_{m,1}\).

4. Create \(m\) additional vertices \(v_{n+1,j}\) for \(j \in [m]\). For all \(j \in [m]\), connect \(v_{n+1,j}\) to \(v_{n,j}\) with two parallel edges and assign \(\text{HW} = 2\) to \(v_{n,j}\).

5. Create \(m\) additional vertices \(v_{0,j}\) for \(j \in [m]\). For all \(j \in [m]\), connect \(v_{0,j}\) to \(v_{1,j}\) with two parallel edges and assign \(\text{HW} = 0\) to \(v_{0,j}\).

It is evident from the construction and the drawing in Figure 6 that \(\Omega_0\) and \(\Omega_1\) are planar. Furthermore, if we enumerate the vertices of the graphs column by column, we obtain linear layouts with cutwidth at most \(n + c\) for a constant \(c \in \mathbb{N}\) independent of the input.

It remains to assign signatures to the vertices \(v_{i,j}\) for \(i \in [n]\) and \(j \in [m]\) constructed in the first step. Recall that \(\phi\) has variables \(x_1, \ldots, x_n\) and clauses \(C_1, \ldots, C_m\). We define an array \(A \in \{0, +, -\}^{n \times m}\) by

\[
A(i, j) = \begin{cases} 
0 & \text{if } x_i \text{ does not appear in } C_j \\
+ & \text{if } x_i \text{ appears in } C_j \\
- & \text{if } \neg x_i \text{ appears in } C_j 
\end{cases}
\]

and assign to \(v_{i,j}\) the signature \(\text{LIT}_A(i,j)\). In the following, we verify that Holant(\(\Omega_0\)) indeed counts the satisfying assignments \(a : \{x_1, \ldots, x_n\} \to \{0, 1\}\) for \(\phi\) with even Hamming weight. The same can be verified for \(\Omega_1\) and the assignments of odd weight. As a notational simplification, for \(i \in [n]\), we write \(v_{i,*}\) for the vertices in row \(i\), and for \(j \in [m]\), we write \(v_{*,j}\) for the vertices in column \(j\).

For each \(i \in [n]\), the signatures \(\text{LIT}_0\), \(\text{LIT}_-\) and \(\text{LIT}_+\) in \(v_{i,*}\) ensure that all horizontal edges between vertices in \(v_{i,*}\) have the same value \(a_i\). By the two columns of \(\text{EVEN}\)
signatures (which act like two vertices of degree $n$ with signature $\text{EVEN}$), evenly many rows have active edges. Altogether, this implies that the satisfying assignments to $\Omega_0$ encode even assignments $a : \{x_1, \ldots, x_n\} \to \{0, 1\}$ to the formula $\varphi$.

For each $j \in \{m\}$, we propagate along the vertical edges in column $v_{\ast, j}$ whether the clause $C_j$ is satisfied by the assignment $a$. For each $1 \leq i \leq n$, the vertex $v_{i, j}$ has two top edges whose assignment encodes whether $C_j$ is satisfied by $a_1, \ldots, a_i$. At $v_{1, j}$, this value is false, as ensured by the $\text{HW}_0$ signatures at $v_{0, j}$. If the variable $x_j$ does not appear in the clause $C_j$, then we propagate the assignment to the top edges downward by definition of $\text{LIT}_0$. If $x_j$ appears positively or negatively, then we check whether $x_i$ satisfies $C_j$ and propagate the result downward by definition of $\text{LIT}_-$ or $\text{LIT}_+$. At the bottom of each column $j \in \{m\}$, the vertex $v_{n+1, j}$ of signature $\text{HW}_{=2}$ tests whether clause $C_j$ was satisfied by the assignment $a = a_1 \ldots a_n$.

Altogether, this shows that the satisfying assignments to $\Omega_0$ encode the satisfying assignments to $\varphi$ where an even number of variables is set to true. A similar proof applies for $\Omega_1$. This proves (5.6) and hence the theorem.

By invoking Theorem 4.1, we obtain hardness for the parameterization by cutwidth.

Proof. of Theorem 1.2. Given a CNF formula $\varphi$ on $n$ variables, invoke Lemma 5.1 to obtain signature graphs $\Omega_0$ and $\Omega_1$ of cutwidth $n + O(1)$ that satisfy (5.6). Using Theorem 4.1 and choosing statement 1, we can then determine $\text{Holant}(\Omega_i)$ for $i \in \{0, 1\}$ with an oracle for unweighted $\text{PERFMATCH}$ that asks only queries on graphs with $n^{O(1)}$ vertices and cutwidth $n + O(1)$. Thus an algorithm with running time $(2 - \epsilon)^{\text{cutw}(G)n^{O(1)}}$ for some $\epsilon > 0$ would yield a $(2 - \epsilon)^{nm^{O(1)}}$ time algorithm for $n$-variable $m$-clause CNF-SAT, violating $\text{SETH}$. As the treewidth of a graph is always bounded from above by its cutwidth, the theorem follows.

5.2 Parameterizing by crossing number. With a slight modification, the construction of Lemma 5.1 also allows us to prove the lower bound for $\text{PERFMATCH}$ parameterized by crossing number. To this end, we first need to observe that the signature $\text{LIT}_0$ can almost be realized by a planar matchgate. That is, we can realize a signature $\text{LIT}_0^*$ that agrees with $\text{LIT}_0$ on assignments $a \in \{0, 1\}^6$ that give the same value to $x_{\text{supp}}$ and $x_{\text{supp}}^2$, but $\text{LIT}_0^*$ may attain arbitrary values on all other assignments.

Lemma 5.2. There is a signature $\text{LIT}_0^*$ that can be realized by a planar matchgate (on edge-weights 1 and $-1$) and satisfies, on all $a \in \{0, 1\}^6$ with $a(x_{\text{supp}}) = a(x_{\text{supp}}^2)$, the condition $\text{LIT}_0(a) = \text{LIT}_0^*(a)$. The signature $\text{LIT}_0^*$ may however attain arbitrary values on all other $a \in \{0, 1\}^6$.

From this, we can conclude the desired lower bound.

Proof. of Theorem 1.7. Since we wish to prove a lower bound under $\text{SETH}$, we may reduce from $\text{SAT}$ on 3-CNF formulas. Given such a formula $\varphi$ on $n$ variables and $m$ clauses, it is clear that $\varphi$ contains at most $3m$ literals that occur positively or negatively.

We construct the planar signature graphs $\Omega_0$ and $\Omega_1$ from $\varphi$ described in Lemma 5.1, replacing each occurrence of $\text{LIT}_0$ by $\text{LIT}_0^*$. It can be checked that this particular replacement preserves Holants: At the top-most vertex $v_{1, j}$ of each column $j \in \{m\}$, the two inputs $x_{\text{supp}}$ and $x_{\text{supp}}^2$ are forced to the same assignment, and by definition of $\text{LIT}_0^*$, $\text{LIT}_-$ and $\text{LIT}_+$, the same applies inductively.

Figure 6: A drawing of $\Omega_0$ with $n = 5$ and $m = 7$. As an example, the column corresponding to clause $C_4 = (x_1 \lor x_3 \lor x_4)$ is highlighted.
at all vertices \(v_{i,j}\) for \(i \in [n]\). Hence, under any satisfying assignment, \(\text{LIT}_0\) would yield the same value as \(\text{LIT}^+\).

As described above, there are at most \(3m\) occurrences of the non-planar signatures \(\text{LIT}_-\) and \(\text{LIT}_+\) in \(\Omega_0\) and \(\Omega_1\). In addition, there are \(O(n + m)\) occurrences of the signatures \(\text{HW}_{-2}\), \(\text{ODD}\), and \(\text{EVEN}\), which in fact have planar matchgates, but we do not use this here. Thus, using Theorem 4.1, we obtain a reduction from \(\#\text{SAT}\) on 3-CNF formulas with \(n\) variables and \(m\) clauses to \(\text{PERF\text{-}MATCH}\) with \(O(nm)\) vertices and crossing number \(O(n + m)\). Consequently, an algorithm for \(\text{PERF\text{-}MATCH}\) with running time \(2^{o(k)}n^{O(1)}\) on graphs with crossing number \(k\) would imply a \(2^{o(nk)}\) time algorithm for \(\#\text{SAT}\) on 3-CNF formulas, and thus refute \(\text{ETH}\) by Lemma 2.2.

6 Parameterizing by cliquewidth

In this section, we prove the following conditional lower bound under \(\text{SETH}\) that complements Theorem 1.3. This lower bound then implies Theorem 1.4.

**Theorem 6.1.** For any fixed \(k \in \mathbb{N}\) and \(\varepsilon > 0\), the following holds: If we can solve \(\text{PERF\text{-}MATCH}\) in time \(O\left(|T| \cdot n^{k-2-\varepsilon}\right)\) on an unweighted graph \(G\) given with an expression for \(G\) with \(k\) large and \(O(1)\) singleton labels, then \(\text{SETH}\) fails.

Since expressions with \(k\) large and \(O(1)\) singleton labels trivially have \(k + O(1)\) labels, this indeed proves Theorem 1.4. We note that it is crucial to obtain unweighted graphs, as the evaluation of PerfMatch is trivially \#P-hard on edge-weighted cliques, whose underlying graphs have cliquewidth 2. (We can simply assign weight zero to non-edges. If zero-weight edges are explicitly forbidden, then a simple interpolation argument allows to simulate them by edges of non-zero weight.)

6.1 The colored hitting set problem. To prove Theorem 6.1, we reduce from counting colorful hitting \(k\)-sets. This is a simple variant of the hitting \(k\)-set problem, for which a tight lower bound was already shown by Patrascu and Williams [75].

**Definition 7.** In the problem \(\text{ColHS}\), the input consists of \(k\) disjoint universes \(X = (X_1, \ldots, X_k)\) with \(n\) individuals each and sets \(A = (A_1, \ldots, A_m)\) with \(A_j \subseteq X_1 \cup \ldots \cup X_k\) for \(j \in [m]\). We may assume that \(X_i = \{i\} \times [n]\) for all \(i \in [k]\). The task is then to determine the number of colorful hitting sets, that is, the size of the set

\[
\text{ColHS}(X,A,k) = \{(s_1, \ldots, s_k) : s_i \in X_i \quad \forall j \in [m] : \exists i \in [k] : s_i \in A_j\}.
\]

The difference to the usual hitting \(k\)-set problem is that the universe \(X\) can be considered to be colored by \(k\) colors, and we are looking for a colorful hitting set. A tight lower bound for this problem can be established in a similar way as in [75].

**Lemma 6.1.** For each fixed \(k \geq 2\), the following holds: If there exists an \(\varepsilon > 0\) such that, for each \(d \in \mathbb{N}\), there is an \(O(n^{k-\varepsilon})\) time algorithm for \(\text{ColHS}\) on instances with universe size \(n\) and \(O(k^d \cdot \log^d n)\) sets, then \(\text{SETH}\) fails. Here, the constant factor in the running time may depend on \(d\).

6.2 Construction of the signature graph. In the remainder of this section, we show how to solve an instance \((X,A,k)\) for the problem \(\text{ColHS}\) by reduction to the problem \(\text{PERF\text{-}MATCH}\) on unweighted graphs of cliquewidth \(k + O(1)\). To this end, we first construct a grid-like signature graph \(\Omega\) on \(k\) rows and \(m\) columns containing cell gates \(C_{i,j}\) for \(i \in [k]\) and \(j \in [m]\). The \(i\)-th row corresponds to the individual from \(X_i\) to be picked in our hitting set, while the \(j\)-th column, which we denote by \(D_j\), corresponds to the set \(A_j\).

The construction resembles that of Section 5: Each row \(i\) propagates an element \(x_i \in X_i\) from left to right, and each column \(D_j\) will check whether \(A_j\) is hit by the elements \(\{x_1, \ldots, x_k\}\) propagated by the rows. The checks at each column \(D_j\) are performed from top to bottom, so that cell \(C_{i,j}\) will know the value of \(x_i\) and whether \(A_j\) was already hit by \(\{x_1, \ldots, x_{i-1}\}\). This allows \(C_{i,j}\) to determine whether \(A_j\) is hit by \(\{x_1, \ldots, x_i\}\) and to propagate this information to the cell \(C_{i+1,j}\). When reaching the bottom-most cell \(C_{k,j}\), the column \(D_j\) will know whether \(A_j\) is hit by \(\{x_1, \ldots, x_k\}\). The main difference to Section 5 is that we do not transfer a Boolean value along the \(i\)-th row, but rather a number \(x_i \in [n]\), which encodes the element \((i, x_i)\) chosen in our hitting set. In particular, cell gates will therefore feature \(O(n)\) dangling edges (rather than a constant number).

In the following, we give a formal construction of a cell by first specifying its underlying graph. We will then attach signatures to this graph to obtain a gate.

**Definition 8.** A cell is a gate containing pairs of dangling edges \((e_{top1}, e_{top2})\) and \((e_{bottom1}, e_{bottom2})\) and \((a_x, b_x)\) for \(x \in [n]\). It is defined as follows: see also Figure 7.

1. Create vertices \(c_1, \ldots, c_n\). For all \(k \in [n]\), connect \(c_k\) to the dangling edges \(a_x\) and \(b_x\).
2. For \(1 \leq k \leq n - 1\), connect \(c_k\) to \(c_{k+1}\) by \(4\) parallel edges. Add \((e_{top1}, e_{top2})\) as dangling edges to \(c_1\) and add \((e_{bottom1}, e_{bottom2})\) to \(c_n\).
3. Connect \(c_1\) to an extra vertex \(u_{top}\) of signature \(\text{HW}_{-2}\) with \(2\) parallel edges, and connect \(c_n\) to an extra vertex \(u_{bottom}\) of signature \(\text{HW}_{-2}\) with \(2\) parallel edges.

Intuitively speaking, the \(4\) parallel edges between consecutive vertices in a cell are used to transfer \(2\) bits in a
parity-consistent way. We require the cell $C_{i,j}$ to behave as follows. For every satisfying assignment $x \in \{0,1\}^{E(C_{i,j})}$ to $C_{i,j}$, the following conditions have to be fulfilled:

**Doubling:** For each pair of dangling edges, its two edges must agree in their values on $x$. This implies in particular that the assignment to $a_1, \ldots, a_n$ is propagated to $b_1, \ldots, b_n$.

**Block:** There exists a number $t \in [n]$ such that all dangling edges $a_\kappa, b_\kappa$ for $1 \leq \kappa \leq t$ are active in $x$, and all dangling edges $a_\kappa, b_\kappa$ for $\kappa > t$ are inactive. That is, the active dangling edges in any satisfying assignment appear in a consecutive block starting at $a_1$ and $b_1$. The length of this block will encode the propagated element from $[n]$.

**Propagation:** If $(e_{top1}, e_{top2})$ is active, then so is $(e_{b_m1}, e_{b_m2})$. This will later ensure that, if a set $A_j$ was already hit by elements $x_1, \ldots, x_{i-1}$ preceding $x_i$, then this information is propagated down. If $(e_{top1}, e_{top2})$ is inactive, then $(e_{b_m1}, e_{b_m2})$ is active iff the set $A_j$ is hit by $(i,t)$, where $t$ is the number defined in the block condition.

It remains to add signatures to the cell $C_{i,j}$ to ensure these conditions. To this end, we define the following “master” signatures $MSTR_0$ and $MSTR_+$ on five pairs of dangling edges. See also Figure 7. There is one “wire” pair, two pairs that we denote as “Boolean” in-/outputs, and two pairs that we denote as “thru” in-/outputs.

These signatures are used in a similar way as the signatures $LIT_0$ and $LIT_+$ from Section 5. If $i \in [k]$, $j \in [m]$ and $\kappa \in [n]$ are such that $(i, \kappa) \in A_j$, then we place $MSTR_+$ at the vertex $c_\kappa$ in the cell $C_{i,j}$. Otherwise, we place $MSTR_0$ at $c_\kappa$.

The main difference to Section 5 is the presence of dangling edges that are considered as “thru” in-/outputs. These will ensure that active dangling edges of the cell $C_{i,j}$ appear in a consecutive block as defined above: An inactive thru input at $c_\kappa$ enforces that the thru output of $c_\kappa$ is inactive as well, and we use this to ensure that the active edges among $a_1, \ldots, a_n$ appear as a consecutive block. Due to the vertex $u_{top}$, the thru inputs at $c_1$ are forced to be active.

The Boolean in-/outputs of $MSTR_0$ and $MSTR_+$ at $c_\kappa$ will then be used to propagate whether the set $A_j$ was hit by any of the elements $(i,1), \ldots, (i, \kappa)$, precisely as for the signatures $LIT_0$ and $LIT_+$ from Section 5: In the case of $MSTR_0$ (which we use if $(i, \kappa) \notin A_j$), we simply propagate the Boolean inputs to the outputs, as seen for $LIT_0$. In the case of $MSTR_+$ (which we use if $(i, \kappa) \in A_j$), we test whether $a_\kappa$ is the first inactive edge after the block of active edges among $a_1, \ldots, a_n$. If this is the case, then $A_j$ is hit by $(i, \kappa)$, and we propagate this information down to the Boolean output pair. This is parallel to the behaviour of $LIT_+$.

In the following, we give formal definitions for the signatures $MSTR_0$ and $MSTR_+$.

**Definition 9.** The master signatures $MSTR_0$ and $MSTR_+$ are 10-ary Boolean signatures

\[ MSTR_0, MSTR_+ : \{0,1\}^{10} \rightarrow \{0,1\} \]

with inputs

\[ (x_1, x_2, (e_{b_m1}^1, e_{b_m2}^1), (e_{b_m1}^2, e_{b_m2}^2), (e_{b_m1}^3, e_{b_m2}^3), (e_{b_m1}^4, e_{b_m2}^4)) \]

They are defined such that $MSTR_0(x) = 1$ and $MSTR_+(x) = 1$ hold for $x \in \{0,1\}^{10}$ iff all of the following conditions are fulfilled:

1. For each pair of dangling edges, such as $(x_1, x_2)$, the two dangling edges in $x$ have the same assignment. In the remainder of this definition, we will therefore refer only to one of the two dangling edges, say to $x_1$.

2. If $e_{b_m1}^{thru}$ is inactive, then so is $e_{b_m2}^{thru}$.

3. If $e_{b_m1}^{thru}$ is active, then $e_{b_m2}^{thru}$ is active iff $x_1$ is.

We apply these signatures as follows to the vertices of a cell $C_{i,j}$: For all $\kappa \in [n]$, if $(i, \kappa) \in A_j$, we place $MSTR_+$ at $c_\kappa$. If $(i, \kappa) \notin A_j$, then we place $MSTR_0$ at $c_\kappa$. This concludes the definition of the cell $C_{i,j}$.

It is clear from the definitions of $MSTR_0$ and $MSTR_+$ that both signatures are even. Furthermore, we can verify that the cell $C_{i,j}$ satisfies the conditions of Definition 8 (doubling, block, propagation): The doubling condition follows from item 1 of Definition 8, the block condition follows from item 2, and the propagation condition follows from item 3.

To proceed, for any $j \in [m]$, we connect the cells $C_{i,j}$ for $i \in [n]$ into the column $D_j$.

**Definition 10.** For any $j \in [m]$, we define the column $D_j$ as follows: First, we create a disjoint union of cells $C_{i,j}, \ldots, C_{k,j}$. Denote the subset of dangling edges $\{a_\kappa \mid \kappa \in [n]\}$ in $C_{i,j}$ by $F_{i,j}^\text{in}$ and the subset $\{b_\kappa \mid \kappa \in [n]\}$ by $F_{i,j}^\text{out}$. This results in $2k$ sets $F_{i,j}^\text{in}$ and $F_{i,j}^\text{out}$ for $i \in [k]$.

Within a column, we then connect cells in the following way:

1. For $1 \leq i \leq n-1$, connect the input pair $(e_{b_m1}^1, e_{b_m2}^1)$ of $C_{i,j}$ with the input pair $(e_{top1}, e_{top2})$ of $C_{i+1,j}$ to obtain two parallel edges.
We observe that the column which has size $n$. Note that the resulting gate $D_j$ has
$$
\{F_{\text{out}}^{k}\}\colon
$$
Columns $D_j$ hits the set $G$ specified in the following.

2. Add two extra vertices $u_0$ and $u_{n+1}$. Assign $\mathbb{HW}_0$ to $u_0$ and $\mathbb{HW}_1=2$ to $u_{n+1}$, similarly as in Lemma 5.1.

Connect $u_0$ with two parallel edges to the input pair $(e_{\text{rep}1},e_{\text{rep}2})$ of $C_{1,j}$, and likewise connect $u_{n+1}$ to the input pair $(e_{\text{rep}1},e_{\text{rep}2})$ of $C_{n,j}$.

Note that the resulting gate $D_j$ has $2kn$ dangling edges, grouped into the $2k$ sets $F_{\text{in}}^{i,j}$ and $F_{\text{out}}^{i,j}$ for $i \in [k]$, each of which has size $n$.

We observe that the column $D_j$ has a Boolean signature when considered as a gate. For $i \in [k]$, the propagation condition of cell $C_{i,j}$ ensures that the assignments for $F_{\text{in}}^{i,j}$ and $F_{\text{in}}^{i,j}$ agree. Furthermore, by construction of the cells $C_{i,j}$ for $i \in [k]$, if $x$ is a satisfying assignment to $D_j$, then there are numbers $t_1,\ldots,t_k \in [n]$ such that precisely the first $t_i$ edges in $F_{\text{in}}^{i,j}$ are active, while no other edges are active. Finally, we can verify that the column $D_j$ yields value $1$ on an input $x$ as above iff the set $\{(i,t_i) \mid i \in [k]\}$ hits the set $A_j$.

To conclude our construction, we connect the $m$ columns $D_1,\ldots,D_m$ to a signature graph $\Omega$. This requires a slightly cumbersome construction to ensure that each colorful hitting $k$-set in $(X,A,k)$ corresponds to a fixed number of satisfying assignments in $G$ while preserving a low cliquewidth. To this end, we “wrap” the cells of columns by bicliques, as demonstrated in Figure 8 and specified in the following.

**Definition 11.** Let $(X,A,k)$ be an instance to $\text{#COLHS}$ with universes $\{i\} \times [n]$ for $i \in [k]$ and sets $A_1,\ldots,A_m$. Without limitation of generality, we may assume that each $A_j$ for $j \in [m]$ contains only elements $(i,\kappa)$ with even $\kappa \in [n]$. Proceed as follows to obtain $\Omega = \Omega(X,A,k)$:

1. For $j \in [m]$, construct the column $D_j$ as in Definition 10. Let $F_{\text{in}}^{i,j}$ and $F_{\text{out}}^{i,j}$ for $i \in [k]$ denote its sets of dangling edges.

2. For each $i \in [k]$ and $j \in [m]$, add four independent sets $I_{i,j}^{(t)}$ of size $n$, for $t \in [4]$, and denote its vertices by $I_{i,j}^{(t)} = \{i^{(t)}_{i,j,1},\ldots,i^{(t)}_{i,j,n}\}$. Assign the signature $\mathbb{HW}_1$ to these vertices.

3. For $\kappa \in [n]$, connect vertex $c_\kappa$ to $a_{i,j,\kappa}^{(1)}$ and to $a_{i,j,\kappa}^{(2)}$.

Then connect $a_{i,j,\kappa}^{(2)}$ to $a_{i,j,\kappa}^{(3)}$.

4. Add all edges between vertices in $I_{j}^{(3)}$ and $I_{j}^{(4)}$ to obtain a complete bipartite graph between these sets. Consider Figure 8 for an example.

5. For $1 \leq j \leq m-1$, add all edges between vertices in $I_{j}^{(4)}$ and $I_{j+1}^{(3)}$, see also Figure 8.

6. Add two vertices $u^*$ and $v^*$.

   a. Delete the independent set $I_{1}^{(1)}$, so that the edges in $F_{\text{in}}^{1,1}\cup\ldots\cup F_{n,1}^{n,1}$ of the left-most column become dangling edges. Connect them to $u^*$ and assign the signature $\text{EVEN}$ to $u^*$.

   b. Delete the independent sets $I_{m,1}^{(2)},I_{m,1}^{(3)},I_{m,1}^{(4)}$, so that the edges in $F_{\text{in}}^{m,1}\cup\ldots\cup F_{n,m}^{n,1}$ of the right-most column become dangling edges. Connect them to $v^*$ and assign the signature $\text{EVEN}$ to $v^*$.

   c. Finally, as in Section 3, replace $u^*$ and $v^*$ by paths of vertices on $\text{EVEN}$ signatures of arity 3.

**Fact 6.1.** For an instance to $\text{#COLHS}$ with $k$ universes of size $n$ each, and $m$ sets, the signature graph $\Omega$ constructed above has $O(kmn)$ vertices. Furthermore, all signatures other than $\mathbb{HW}_0$ have constant arity.

**6.3 Correctness of the signature graph.** We prove that, for $\Omega = \Omega(X,A,k)$, the quantity $\text{Holant}(\Omega)$ counts colorful hitting sets in $(X,A,k)$ up to a simple factor.

**Lemma 6.2.** Let $(X,A,k)$ be an instance to $\text{#COLHS}$ with $X_i = \{i\} \times [n]$ for $i \in [k]$ and sets $A_1,\ldots,A_m$. Let $\Omega = \Omega(X,A)$ be constructed as in Definition 11. Then $\text{Holant}(\Omega) = (n!)^k (m-1) \cdot \text{#COLHS}(X,A,k)$.

Proof. Consider the satisfying assignments for $\Omega$. It follows from Definition 8 that, for each $i \in [k]$, there exists
a number $x_i \in [n]$ such that the first $x_i$ dangling edges in $F_{i,1}^{in}$ and $F_{i,1}^{out}$ are active, while no other edges in these sets are active. Please recall Figure 8 in the following.

- Since the first $x_i$ dangling edges in $F_{i,1}^{in}$ and $F_{i,1}^{out}$ are active, while no other dangling edges are active, the following holds: Among the independent set $I_{i,1}^{(2)}$, precisely the first $x_i$ vertices are matched by dangling edges of the cell $C_{i,j}$.
- This in turn implies that exactly the last $n - x_i$ edges are active between $I_{i,1}^{(2)}$ and $I_{i,1}^{(3)}$, so the last $n - x_i$ vertices of $I_{i,1}^{(3)}$ are matched by edges from $I_{i,1}^{(2)}$ to $I_{i,1}^{(3)}$.
- Consequently, there are $x_i$ active edges between $I_{i,1}^{(4)}$ and $I_{i,1}^{(3)}$. Since their endpoints in $I_{i,1}^{(3)}$ are fixed (to the first $x_i$ vertices, as the last $n - x_i$ vertices are already matched), there are $(n)!$ ways of choosing these active edges. Here, we denote by $(n)!$ the falling factorial $(n)_i = (n)(n-1)\ldots(n-t+1).$
- Finally, there are $n - x_i$ active edges between $I_{i,2}^{(1)}$ and $I_{i,2}^{(4)}$. The endpoints of these $n - x_i$ edges in $I_{i,2}^{(4)}$ are fixed by the choices in the preceding step, and by the block property of the cell $C_{i,2}$, the endpoints of these edges in $I_{i,2}^{(1)}$ are fixed to the last $n - x_i$ vertices as well. Hence, there are $(n-x_i)!$ ways of choosing the active edges between $I_{i,1}^{(4)}$ and $I_{i,2}^{(1)}$. In particular, the first $x_i$ edges among $F_{i,2}^{in}$ are now active. That is, the value $x_i$ was propagated correctly from $C_{i,1}$ to $C_{i,2}$.

From this initial step, we obtain inductively that, for satisfying assignment and every $i \in [k]$ and $j \in [m]$, the first $x_i$ edges in every set $F_{i,j}^{in}$ and $F_{i,j}^{out}$ are active. We can thus associate with each satisfying assignment $x$ to $\Omega$ the encoding of a set $S_x = \{(i,x_i) \mid i \in [k]\}$ with $|S_x \cap X_i| = 1$ for all $i \in [k]$.

For a given set $S$ as above, either there is no satisfying assignment $x$ to $\Omega$ with $S = S_x$ (this occurs when at least one of the column signatures $D_j$ yields value 0, which in turn occurs precisely if there is some set $A_j$ that was not hit by $S$), or there are precisely

$$\prod_{i \in [k]} ((n-x_i)!)^{m-1} = (n!)^{k(m-1)}$$

satisfying assignments $x$ to $\Omega$, each having value 1 as all involved signatures are Boolean. We use here that $\sum n_i$ is even to ensure that an even number of dangling edges is active in each column, thus satisfying the paths of EVEN signatures at the first and last columns. This proves the lemma.

### 6.4 Bounding the cliquewidth

Finally, we bound the cliquewidth of the graph $\Omega = \Omega(X,A,k)$ obtained from an instance $(X,A,k)$ of counting colorful hitting $k$-sets via Definition 11. Let us note that parallel edges (which appear in $\Omega$) cannot be obtained by clique expressions. We can however subdivide each parallel edge twice, and this way obtain a simple graph $\Omega'$ from $\Omega$. If all parallel edges are singular (which can be verified for $\Omega$), then the subdivision introduces only $O(1)$ new singleton labels.

**Lemma 6.3.** Let $(X,A,k)$ be an instance for $\text{#ColHS}$ and let $\Omega = \Omega(X,A,k)$. Then we can construct in polynomial time a $(k+2,O(1))$-expression for $\Omega$.

**Proof.** (Sketch.) We construct the graph $\Omega$ column by column. For this, we use a large forget label $\kappa^*$, large labels $\kappa_1, \ldots, \kappa_k$, and an additional large working label $\tau$. Furthermore, we use $O(1)$ singleton labels.

To construct column $D_j$, assume that the columns $D_1, \ldots, D_{j-1}$ have been constructed already, that $\tau$ is empty, and that the label $\kappa_i$ for $i \in [k]$ contains precisely the elements in the independent set $I_{i,j-1}^{(3)}$ specified in Definition 11. Then we can construct $D_j$ and its independent sets $I_{i,j}^{(b)}$ for $i \in [k]$ and $b \in [4]$ as follows. Please consider again Figure 8.

Consider $i \in [k]$ fixed for now. For $b \in [4]$, let us write $I_{i,j}^{(b)} = \{a_{i,j,k}^{(b)} \mid k \in [n]\}$. For each $k \in [n]$, we perform the following steps:

![Figure 8: The subgraph of $G$ containing the cell $C_{i,j}$ and its surrounding independent sets](insert figure here)
1. Construct the vertices $a_{i,j,k}^{(1)}$, $a_{i,j,k}^{(2)}$, and $a_{i,j,k}^{(3)}$, and the cell vertex $c_k$ of $C_{i,j}$ on singleton labels.

2. Connect $a_{i,j,k}^{(1)}$ to all vertices of label $\kappa_i$ by means of the join operation. This adds all edges between $a_{i,j,k}^{(1)}$ and the independent set $I_{i,j}^{(4)}$.

3. Connect $c_k$ to $a_{i,j,k}^{(2)}$ and $a_{i,j,k}^{(3)}$, and connect $a_{i,j,k}^{(2)}$ to $a_{i,j,k}^{(3)}$ by means of join operations on their singleton labels.

4. Relabel $a_{i,j,k}^{(3)}$ from its singleton label to $\tau$. Relabel $a_{i,j,k}^{(1)}$ and $a_{i,j,k}^{(2)}$ to the forget label $\kappa^\ast$.

5. If $\kappa > 1$, connect $c_{\kappa-1}$ to $c_k$. (Here, we assume that $c_{\kappa-1}$ is contained in a singleton label.) Relabel $c_{\kappa-1}$ to the forget label $\kappa^\ast$.

So far, we have constructed the cell $C_{i,j}$ and the independent sets $I_{i,j}^{(1)}$, $I_{i,j}^{(2)}$, and $I_{i,j}^{(3)}$. Our expression contains precisely the independent set $I_{i,j}^{(3)}$ in label $\tau$. To add the independent set $I_{i,j}^{(4)}$, we proceed as follows:

1. Relabel $\kappa_i$ to the forget label $\kappa^\ast$. Note that $\kappa_i$ is empty now.

2. Add the independent set $I_{i,j}^{(4)}$ on label $\kappa_i$.

3. Join labels $\tau$ and $\kappa_i$ so as to add all edges between $I_{i,j}^{(3)}$ and $I_{i,j}^{(4)}$.

4. Relabel $\tau$ to the forget label $\kappa^\ast$.

This finishes the construction of a single cell $C_{i,j}$ and its independent sets $I_{i,j}^{(b)}$ for $i \in [k]$ and $b \in [4]$. Repeat this process for all $i \in [k]$, and connect vertically adjacent cells $C_{i,j}$ and $C_{i+1,j}$ to columns by adding edges between vertices on singleton labels.

6.5 Finishing the proof. By combining Lemmas 6.1, 6.2 and 6.3, we obtain the proof of Theorem 6.1.

Proof. By Lemma 6.1, an algorithm with running time $O(n^{k+\varepsilon})$ for $\#\text{Col.HS}$ on instances $(X,A,k)$ with $k$ universes of size $n$ and $m = O(\log^d(n))$ sets for fixed $d \in \mathbb{N}$ would refute $\#$SETH. Using Definition 11, we can transform $(X,A,k)$ to a signature graph $\Omega$ such that

1. the value $\text{Holant}(\Omega)$ allows to recover the solution $\#\text{Col.HS}(X,A,k)$ by Lemma 6.2.
2. all signatures appearing in $\Omega$ other than $\mathbb{H}^{\mathbb{W}=1}$ are even by Fact 6.1.
3. $\Omega$ has $O(nmk) = O(n\log^d n)$ vertices by Fact 6.1.
4. and there is an expression for $\Omega$ with at most $k+2$ large labels and $O(1)$ singleton labels by Lemma 6.3. In particular, all vertices of signatures other than $\mathbb{H}^{\mathbb{W}=1}$ are singular in this expression.

Together with Theorem 4.1, we then obtain a reduction from $\text{Holant}(\Omega)$ to $\#\text{PERFMatch}$ on graphs $G'$ that admit clique expressions with $k+2$ large and $O(1)$ labels. The theorem also asserts that the size of these graphs is bounded by $O(n\log^{d+2} n)$. Hence, if there were an algorithm with running time $O((k^{2-\varepsilon})$ for solving $\#\text{PERFMatch}$ on $t$-vertex graphs $G'$ that are given together with a $(k, O(1))$-expression, then we would obtain an $O((n\log^{d+2} n)^{k-\varepsilon}) = O(n^{k-\varepsilon})$ time algorithm for solving $\#\text{Col.HS}$, for any $\varepsilon' < \varepsilon$, thus refuting $\#$SETH.

References


